自顶语法分析

2020年7月1日 12:22

1. 自顶向下的分析(Top-Down Parsing)
    a. 从分析树的顶部（根节点）向底部（叶节点）方向构造分析树
    b. 可以看成是从文法开始符号S推导出词串w的过程
    c. 每一步推导中，都需要做两个选择
        i. 替换当前句型中的**哪个非终结符**
            1) 最左推导(Left-most Derivation)：总是选择每个句型的最左非终结符进行替换，记作S=>^*_lm α，称 α 是当前文法的最左句型(left-sentential form)，其逆过程是最右规约
            2) 最右推导(Right-most Derivation)同理，称为规范推导，其逆过程是最左规约
            3) 在自底向上的分析中，总是采用最左归约的方式，因此把最左归约称为规范归约，而最右推导相应地称为规范推导。自顶向下也常用最左，但最左推导似乎没有规范推导的说法
        ii. 用该非终结符的**哪个候选式**进行替换
    d. 分析的通用形式：递归下降分析(Recursive-Descent Parsing)
        i. 由一组过程组成，每个过程对应一个非终结符
        ii. 从文法开始符号S对应的过程开始，其中递归调用文法中其它非终结符对应的过程。如果S对应的过程体恰好扫描了整个输入串，则成功完成语法分析
        iii. 尝试失败时可能需要回溯(backtracking)，导致效率较低，称需要回溯的分析为不确定的分析
    e. 预测分析(Predictive Parsing)：是递归下降分析技术的一个特例，通过在输入中向前看固定个数（通常是一个）符号来选择正确的A-产生式
        i. 对某些文法构造出向前看k个输入符号的预测分析器，称为LL(k) 文法类
        ii. 不需要回溯，是一种确定的自顶向下分析方法
2. 文法转换
    a. 不能自顶向下的两种文法
        i. 如果一个文法中有一个非终结符 A 使得对某个串α存在一个推导A =>^+ Aα ，那么这个文法就是**左递归**的，会使递归下降分析器陷入无限循环
            1) 含有A→Aα形式产生式的文法称为是直接左递归的(immediate left recursive)
            2) 经过两步或两步以上推导产生的左递归称为是间接左递归的
        ii. 同一非终结符的多个候选式存在**共同前缀**，将导致回溯现象
    b. 消除左递归
        i. 消除直接左递归：将左递归换成右递归。 ：将左递归换成右递归
            1) 例：A→ Aα | β(α≠ ε,β不以A开头)，其实就是r=βα*，可转换成A→ β A′和A′ → α A′ | ε

2) 推广：A → A α1 | A α2 | … | A αn | β1 | β2 | … | βm (αi≠ ε,βj不以A开头)可转化成：A→ β1 A′ | β2 A′ | … | βmA′ 和 A′→α1 A′ | α2 A′ | … | αnA′ | ε
3) 代价：引进了新非终结符和ε_产生式（形如A→ε）

ii. 消除间接左递归：将前面的产生式代入进后面的产生式，使后面的产生式变成直接左递归，再转换，即：

1) 输入：不含循环推导（即形如A=>^+ A的推导）和ε-产生式的文法G
2) 输出：等价的无左递归文法
3) 方法：

```
按照某个顺序将非终结符号排序为A1，A2，…，An.
for( 从1到n的每个r ) {
        for( 从1到r -1的每个l ) {
                将每个形如Ar→ Alγ的产生式替换为产生式组Ar→δ 1γ|δ2γ|…|δkγ， 其中Al→δ 1|δ2|… |δk，是所有的Al产生式
        }
        消除Ar产生式之间的立即左递归
}
```

c. 消除公共前缀：提取左公因子(left factoring)

i. 推迟决定，等读入了足够的输入，获得足够信息后再做出正确的选择。例：将产生式形如S→aA|aB改写成：S→aS' 和 S'→A|B，即：

1) 输入：文法G
2) 输出：等价的提取了左公因子的文法
3) 方法：

对每个非终结符A，找出它的两个或多个选项之间的最长公共前缀α。如果α ≠ ε， 即存在一个非平凡的(nontrivial)公共前缀，那么将所有A-产生式:
A → αβ1 | αβ2 | … | αβn| γ1| γ2| … | γm
替换为
A → αA'| γ1| γ2| … | γm 和 A'→ β1| β2| … | βn
其中，γi表示所有不以α开头的产生式体；A'是一个新的非终结符。不断应用这个转换，直到每个非终结符的任意两个产生式体都没有公共前缀为止

3. 可预测分析的文法

a. 预测分析法的工作过程：从文法开始符号出发，在每一步推导过程中根据当前句型的最左非终结符A和当前输入符号a，选择正确的A-产生式。为保证分析的确定性，选出的候选式必须是唯一的

b. S_文法（简单的确定性文法，Korenjak& Hopcroft，1966）

i. 每个产生式的右部都以终结符开始
   1) S_文法不允许ε产生式（ε不是终结符）。ε产生式的使用见d.ii

ii. 同一非终结符的各个候选式的首终结符都不同

c. 非终结符A的后继符号集FOLLOW(A)

i. 可能在某个句型中**紧跟在A后边的终结符**a的集合，记为FOLLOW(A)
FOLLOW(A)={**a**|S=>*αA**a**β, **a**∈V_T，α,β∈(V_T∪V_N)*}（紧跟在A后边是指A结束了，后面的内容）

ii. 特例：如果A是某个句型的的最右符号， 则将结束符“$”添加到FOLLOW(A)中

d. 产生式 A→β 的可选集SELECT(A→β)

i. 产生式A→β的可选集是指**可以选用该产生式进行推导时对应的输入符号**的集

合，记为SELECT(A→β)

ii. 例：ε产生式的SELECT(A→ε)=FOLLOW(A)，即当前某非终结符A与当前输入符a不匹配时，若存在 A→ε，可通过检查a是否在FOLLOW(A)来决定是否使用产生式A→ε，（若文法中无A→ε，则应报错）

e. q_文法

i. 每个产生式的右部或为ε，或以终结符开始

1) q_文法不允许右部以非终结符打头的产生式。非终结符开头见f.iii

ii. 具有相同左部的产生式有不相交的可选集

f. 符号串α的串首终结符集FIRST(α)

i. 串首终结符/首终结符：串首第一个符号，并且是终结符

ii. 对任意α∈(V_T∪V_N)^+, FIRST(α)={ a| α =>^*aβ，a∈V_T，β∈(V_T∪V_N)*}；如果α =>^*ε，那么ε∈FIRST(α)。即给定一个文法符号串α，α的串首终结符集FIRST(α)被为：**可从α推导出的所有串首终结符**构成的集合；特例，如果α=>^*ε， 那么ε也在FIRST(α)

iii. 产生式A→α的可选集SELECT：

1) if(!ε∈FIRST(α)) SELECT(A→α)=FIRST(α)

2) if(ε∈FIRST(α)) SELECT(A→α)=(FIRST(α)-{ε})∪FOLLOW(A)

g. LL(1)文法：从左开始扫描，最左推导，向前多看一步就能预测

i. 文法G是LL(1)的，当且仅当G的任两个具有相同左部的产生式A → α | β 满足下面的条件：

1) 若α和β都不能推导出 ε，则FIRST (α) ∩ FIRST (β) = 空集Φ，或者说，不存在终结符a使得α和β都能够推导出以a开头的串

2) α和β至多有一个能推导出ε

a) 否则α和β的产生式的可选集都有FOLLOW(A)，就相交了

3) 如果α =>^* ε，则FIRST (β) ∩ FOLLOW(A) = 空集Φ；

4) 如果β =>^* ε，则FIRST (α) ∩ FOLLOW(A) = 空集Φ；

ii. 简单说就是**同一非终结符的各个产生式的可选集互不相交**

4. 实现算法

a. 求符号X的串首终结符集FIRST(X)：不断应用下列规则，直到没有新的终结符或ε可以被加入到任何FIRST(X)集合中为止：

i. if (X是一个终结符)

FIRST（X）= { X };

ii. if (X是一个非终结符 AND X→Y1…Yk∈P(k≥1)) /* 即Y是非空串 */

1) if (存在某个i使得a在FIRST (Yi)中 AND ε 在所有的 FIRST(Y1) , … , FIRST(Yi-1)中) /* 即Y的前缀子串Y1…Yi-1=>^* ε */

FIRST(X).insert(a);

2) if (所有的j= 1,2, . . . , k，ε在 FIRST(Yj)中)

FIRST(X).insert(ε);

iii. if (X→ε∈P)

FIRST(X).insert(ε);

b. 求串X1X2……XN的FIRST：
    i. FIRST(X1X2…Xn).insert(FIRST(X1)中所有的非ε符号);
    ii. for(int i=0; ε在FIRST(X1X2……Xi-1)中; ++i)
        FIRST(X1X2…Xn).insert(FIRST(Xi)中的所有非ε符号);
    iii. if (对所有的i，ε都在FIRST(Xi)中)
        FIRST(X1X2…Xn).insert(ε);
c. 计算非终结符A的FOLLOW(A)：不断应用下列规则，直到没有新的终结符可以被加入到任何FOLLOW集合中为止：
    i. FOLLOW(S).insert($); //S是开始符号，$是输入右端的结束标记
    ii. if(存在产生式A→αBβ)
        FOLLOW(B).insert(FIRST(β)中除ε之外的所有符号);
    iii. if(存在产生式A→αB OR 存在产生式A→αBβ AND FIRST(β)包含ε)
        FOLLOW(B).insert(FOLLOW(A)的所有符号);
d. 预测分析步骤：
    i. 构造文法
    ii. 改造文法：消除二义性，消除左递归，消除回溯
    iii. 求各变量的FIRST和FOLLOW，进而求的SELECT
    iv. 检查是不是LL(1)，若是则构造预测分析表
    v. 按下一节内容写分析算法

5. 递归的预测分析
a. 预测分析表：每行对应一个非终结符，每列对应一个输入符号，每个元素是该非终结符左部和该输入符号所对应的的产生式（产生式的SELECT集里的符号即为每列的符号）
b. 递归的预测分析法：在递归下降分析中，编写每个非终结符对应的过程时，根据预测分析表进行产生式的选择。具体地说，每个左部非终结符对应的过程：遍历分析表内的该行，用if else根据该行每个非空元素所在列，决定对应产生式，遍历该产生式，
    i. 遇到终结符时，若下一个token不是该终结符，则报错
    ii. 遇到非终结符时，调用下一个token对应的非终结符所对应的过程
c. 与非递归的比较
    i. 优点：直观，不需载入分析表
    ii. 缺点：程序规模大，效率低，自动生成难

6. 非递归的预测分析
a. 不为每个非终结符编写递归下降过程，而是根据预测分析表构造自动机，因此也称为表驱动的预测分析
b. 该自动机在有穷自动机的基础上使用了栈，用于记录之前读取到的符号数量，称为Push Down Automata下推自动机PDA
    i. 初始时，栈里只有文法开始符和结束符$
    ii. 每次根据输入元素和栈顶元素，选择预测分析表内的产生式，弹出栈顶，压入产生式（从产生式体右开始压，栈顶应为产生式体首符）

iii. 产生式右部是ε时相当于是直接弹出栈顶元素

iv. 当栈顶是终结符，且和输入符匹配，就双双出栈（不匹配就是出错了）

c. 输入：一个串w和文法G的分析表M

d. 输出：如果w在L(G)中，输出w的最左推导；否则给出错误指示

e. 方法：最初，语法分析器的格局如下：输入缓冲区中是w$，G的开始符号位于栈顶，其下面是$。下面的程序使用预测分析表M生成了处理这个输入的预测分析过程

```
设置ip使它指向w的第一个符号，其中ip是输入指针；
令X=栈顶符号；
while ( X ≠ $) { /* 栈非空*/
        if( X 等于ip所指向的符号a ) 栈.弹出操作，将ip向前移动一个位置；
        else if ( X是一个终结符号) error( ) ;
        else if ( M[X，a]为空) error( ) ;
        else if ( M[X，a] = X →Y1Y2… Yk) {
                输出产生式X →Y1Y2… Yk;
                将Yk，Yk-1…，Y1压入栈中，其中Y1位于栈顶；
                弹出栈顶符号；
        }
        令X=栈顶符号
}
```

7. 预测分析错误处理

a. 出错的时机

i. 栈顶终结符不匹配当前输入符

ii. 栈顶非终结符与当前输入符在预测分析表内无对应产生式

b. 错误恢复方式

i. 恐慌模式：忽略输入中的一些符号，直到输入中出现合法单元

ii. 改进的恐慌模式：忽略一些符号，直到出现由设计者选定的同步词法单元(synchronizing token)集合中的某个词法单元

1) 其效果依赖于同步集合的选取。集合的选取应该使得语法分析器能从实际遇到的错误中快速恢复

2) 例如可以把FOLLOW(A)中的所有终结符放入非终结符A的同步记号集合。即在分析表内给A行FOLLOW(A)列元素们设为synch，读到synch则恐慌弹出栈顶，即跳过A，直接分析新栈顶和FOLLOW(A)是否匹配

iii. 若终结符在栈顶而不能匹配，可以直接弹出该终结符